System Design Document for uDoDo

Hanna Materne, Josefin Ondrus,
Sofia Edström, Emma Gustafsson

Objektorienterat Programmeringsprojekt, TDA367
Chalmers Tekniska Högskola

2013-05-26

Table of Contents

Innehållsförteckning

1 Introduction **3**
1.1 Design goals **3**
1.2 Definitions, acronyms and abbreviations **3**
2 System design **3**
2.1 Overview **3**
2.1.1 The construction of the view-package 3
2.1.2 The construction of the controller-package 3
2.1.3 The construction of the model-package 4
2.1.3 The construction of the utility-package 4
2.1.5 Event handling 4
2.2.1 General 4
2.2.3 Layering 5
2.3 Concurrency issues **6**
2.4 Persistent data management **6**
2.5 Access control and security **6**
2.6 Boundary conditions **7**
3 References **7**
APPENDIX **8**

Version: 1.0

Date: 2013-05-26

Author: Hanna Materne, Emma Gustafsson, Josefin Ondrus, Sofia Edström

This version overrides all previous versions.

1 Introduction

This chapter will provide an overview of the project.

1.1 Design goals

The design of the project must be loosely coupled, this to make the design testable. It should be possible to isolate parts for test.

One of the goals with the project is to provide code that is easy to modify, but also easy for the developers to add new functionality. This will make it possible for the developers to continue the progress even after the course.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface.
- Java, platform independent programming language
- JRE, the Java Run time Environment. Additional software needed to run an Java application.
- MVC, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data.
- Task, something that user needs to do, presented in the todolist.
- Todolist, a list containing tasks.
- Category, a way to divide tasks into different classifications.

2 System design

2.1 Overview

The application will use a MVC model. This to separate parts from each other. Controllerclasses will handle almost all the logic, modelclasses will handle data and the viewclasses will handle everything concerning the GUI.

2.1.1 The construction of the view-package

To be able to control the visuality of one panel at each time, the class GUIView figures as a main frame where all separate views are added after user interaction. The classes HeaderView, ListView, CategoryView, TopView, CategoryListView and TaskSettingView are all added into GUIView at launch of the application.

2.1.2 The construction of the controller-package

Each controller class holds functions concerning ActionListener that is needed in the application. The controllers are set to each model class that needs connection to a controller.

CategoryPanelController is the controller class handling each separate CategoryModel, while the CategoryController controls the adding of new categories. Main is the class used for running the application.

2.1.3 The construction of the model-package

The model classes is used to represent the different components in the program. Every task is represented logically by a TaskModel, as is every category represented by a CategoryModel.

2.1.3 The construction of the utility-package

In the utility package there are classes for simple grahpical changes such as changing color and font of of the application. The functionality of saving and reloading the todolist lies in Save and Read. A simple class for adding the saved panels is the FileToProgramclass. Unfortunatly, this function is not completely implemented by deadline and therefore not working properly. The logic in FileToProgram is not at all close to how proper I/O is used, but sadly, we did not have a lot of experience regarding this and did not have enough time to explore it further.

2.1.5 Event handling

All events in the application is either handled by MouseListener or ActionListener, and its logic is contained in the controllerclasses. No other event handling was found needed, as uDoDo is not operated by multiple users.

All editable panels have MouseListener both for marking out which panel is selected with hover and if a panel is clicked. ActionListeners are used for all buttons contained in the program.

2.2 Software decomposition

2.2.1 General

Following modules is the application decomposed into, see figure 1:

- Main, class that is holding main-method.
- viewpackage, several viewclasses, mostly holding one panel, that adds into GUI-VIEW which is main GUI for applivation.
- controllerpackage, holds all controllerclasses that connects modelclasses with viewclasses.
- modelpackage, holds all modelclasses, containing data.
- calendar, package holding imported classes for displaying a calendar for the user to select deadline of a task from.
- utility, contains I/O (for filehandling) and some other general utilities.

2.2.2 Decomposition into subsystems

The package calendar contains JCalendar imported from programmersheaven.com at the 15th of May 2013. The author for the classes is Niraj Agarwal. The JCalendar is preferred since uDoDo needed a graphical user interaction to set a date as

deadline for a task. It has graphical resemblance of swingcomponents, which was considered appropriate since uDoDo is constructed in swing.
Another subsystem is the file handling in package utility (not a unified subsystem, just classes handling io).

2.2.3 Layering

Figure 1 shows the layering. The higher layers are at the top of the figure.

2.2.4 Dependency analysis

In the figure below, dependencies are shown. There are unfortunately some dependencies, but only one calling for another.

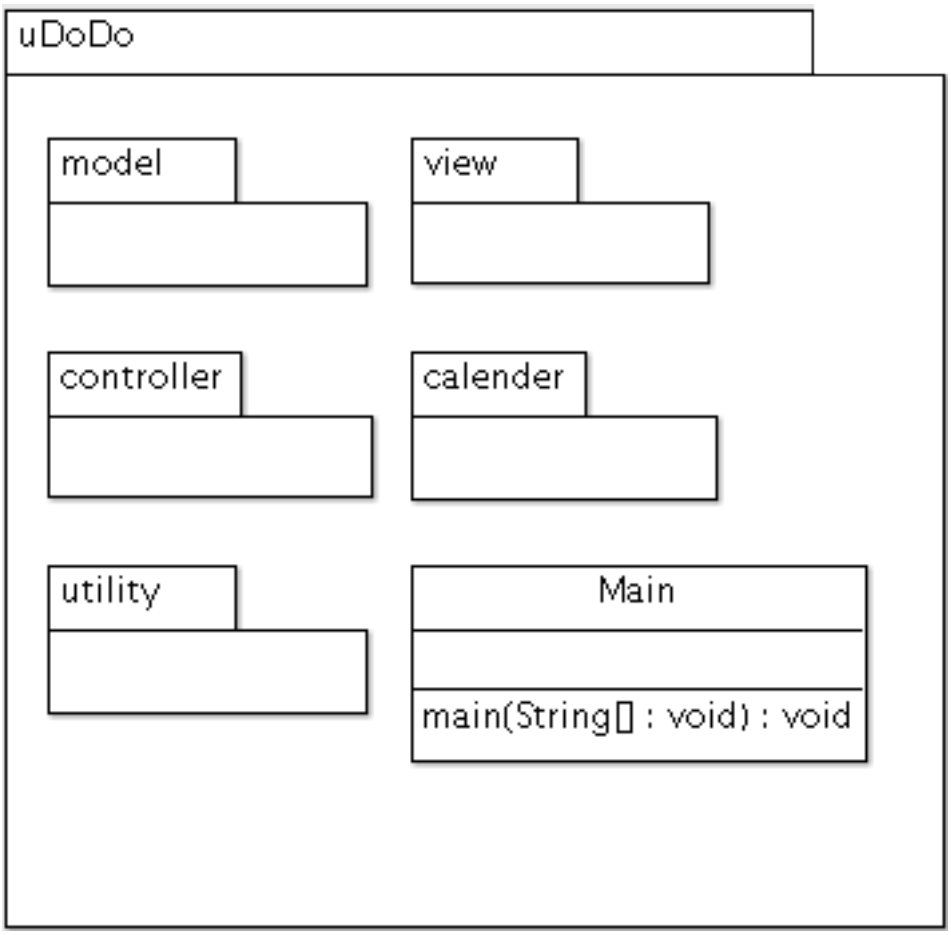


Figure 1: High level design

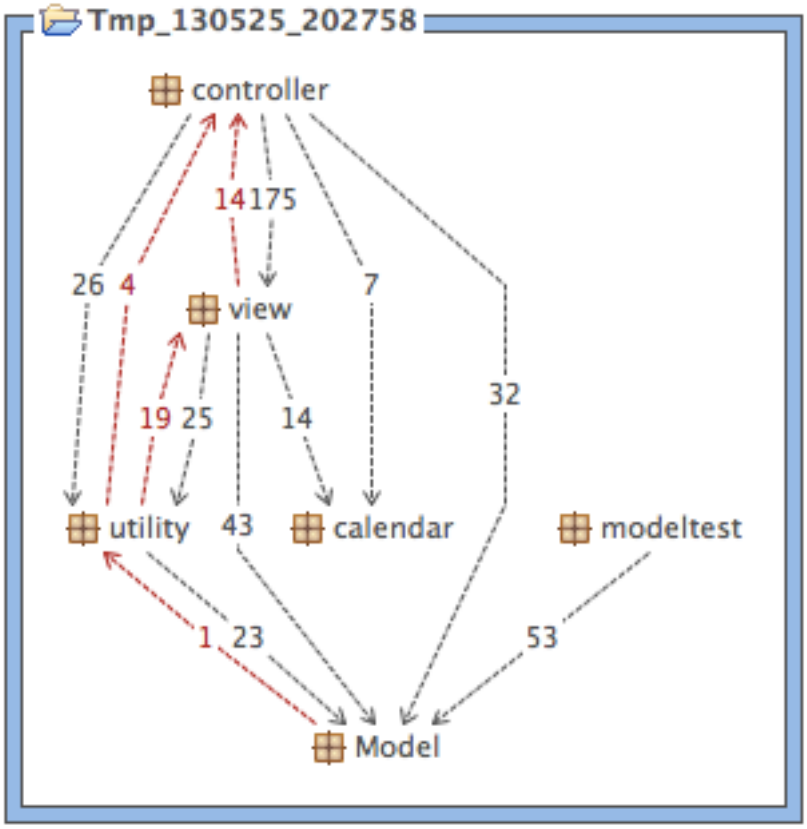


Figure 2: Layering and Dependency analysis

2.3 Concurrency issues

NA. This is a single threaded application. Swing event thread will handle the application. If possible increased response, there could be background threads. Though, this will not raise any concurrency issues.

2.4 Persistent data management

All persistent data is ment to be saved in .ser files locally in eclipse, but is unfortunatly not working correctly by deadline (see chapter 2.1.3 The construction of the utility-package).

2.5 Access control and security

NA

2.6 Boundary conditions

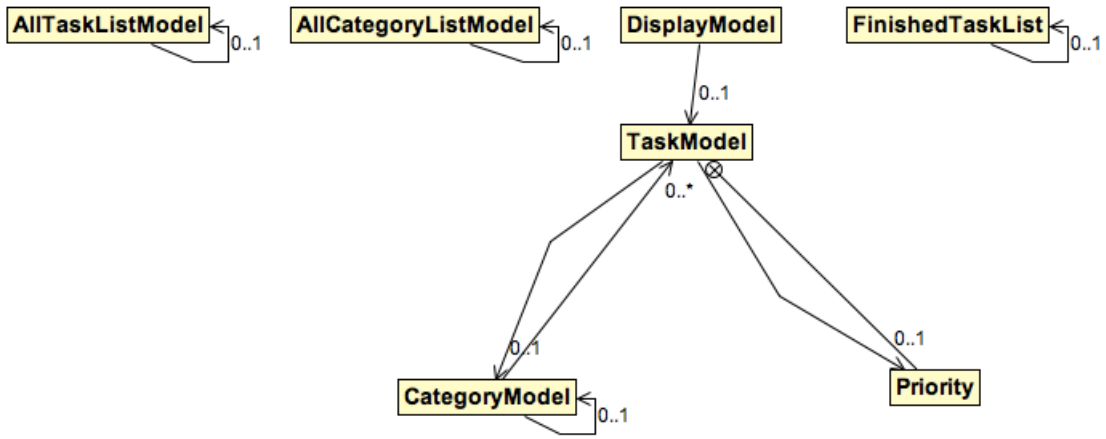
NA. Application launched and exited as normal desktop application (scripts).

3 References

1. MVC, see http://en.wikipedia.org/wiki/Model-View-Controller
2. JCalendar, see http://programmersheaven.com/download/36902/download.aspx

APPENDIX

Class diagrams for packages



Figur NUMBER : ClassDiagram for the modelpackage